

Code Assessment of the LZ Deploy Smart Contracts

October 21, 2026

Produced for



Contents

1	Executive Summary	3
2	Assessment Overview	4
3	System Overview	5
4	Trust Model	7
5	System Considerations	9
6	Terminology	11
7	Findings	12
8	Limitations and use of report	14

1 Executive Summary

Dear Sky Team,

Thank you for trusting ChainSecurity with the review of LZ Deploy, the deployer contracts of the Sky LayerZero bridges. As a globally trusted security partner for tier-one DeFi protocols and cross-chain infrastructure, ChainSecurity provides specialized, independent technical analysis of complex smart contract systems. We greatly appreciate the opportunity to continue our collaboration with Sky on the next generation of SkyLink.

Sky implements LZ Deploy, five deployer contracts that deploy the OApps of its token, governance and SSR bridges, configure them to the exact state the onboarding spells of `lz-init-lib` expect, and hand them to governance.

We have conducted a thorough review of the scope. The most important areas covered in our review are:

- Access control at handoff, since privileges left with the deploying account or with a deployer contract might become a backdoor.
- Functional correctness of the configuration, since a route left on default libraries, on a default DVN set, or with non-zero rate limits would be exposed before governance activates it.
- The two-step deployment of the SSR bridge, since it relies on predicting the receiver's address before the receiver exists.

We provide this audit report to assist in forming an opinion on the risk profile of LZ Deploy. In summary, we find that the codebase provides a good level of security, if operated under the assumptions documented in [System Overview](#), [Trust Model](#) and [System Considerations](#). Note that the gas values of the example scripts are still too low for the Amsterdam hard fork, see [Gas Values After the Amsterdam Hard Fork](#).

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the LZ Deploy repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

Version	Date	Commit Hash	Note
1	7 October 2026	b295d5c9f806d3c510edc67fcc0443908fd4a010	Initial review
2	21 October 2026	e0b2b1690c593a01852705661210bfb761445fae	Fixes

For the Solidity smart contracts, the compiler version `0.8.24` was chosen with the EVM version set to `cancun`.

The following files are in scope:

```
src/
├── L10FTDeployer.sol
├── L1SsrBridgeDeployer.sol
├── L2GovBridgeDeployer.sol
├── L20FTDeployer.sol
└── L2SsrBridgeDeployer.sol
```

2.1.1 Excluded from scope

Any other files are out of scope, in particular:

- Scripts and tests.
- All dependencies, including the contracts the deployers instantiate and configure (`sky-oapp-oft`, `lz-governance-relay`, `lz-init-lib` and `xchain-ssr-oracle`), LayerZero and OpenZeppelin. They are assumed to work correctly as documented.
- The DVN infrastructure, the bridged tokens and their mint and burn authority, and the SSR rate-provider adapters.
- The governance spells that onboard the deployments, and the correctness of the parameters supplied to the deployers.

3 System Overview

This system overview describes the initially received version (`Version 1`) of the contracts as defined in the [Assessment Overview](#). At the end of this report section, we have added subsections for each of the changes according to the versions.

Sky bridges USDS and sUSDS, governance actions, and the sUSDS savings rate (SSR) between Ethereum mainnet and other chains using LayerZero. LZ Deploy deploys the contracts of these bridges, configures them, and hands them to governance. It consists of five deployer contracts, each deploying one part of the system. The governance spells built with `lz-init-lib` then verify and activate the deployments.

3.1 Governance Bridging

`L2GovBridgeDeployer` deploys a remote chain's half of the governance bridge. The mainnet side already exists and is wired to the new chain by a governance spell.

- It deploys a `Governance0AppReceiver` peered with the given mainnet `Governance0AppSender` (chainlog `LZ_GOV_SENDER`), and sets its receive library and ULN configuration for mainnet.
- It deploys an `L2GovernanceRelay` for messages from the given `L1GovernanceRelay` (chainlog `LZ_GOV_RELAY`), with the given `delay` , `gracePeriod` and guardians (`bud`).
- It makes the relay the receiver's delegate and owner. The relay itself has no owner and changes its parameters only through the governance actions it executes.

3.2 Token Bridging

`L10FTDeployer` deploys the mainnet lockbox (`Sky0FTAdapter`) for one token, and `L20FTDeployer` the adapter on a remote chain (`Sky0FTAdapterMintBurn`). Both perform the following steps:

- Deploy the adapter behind an `ERC1967Proxy` , with the deployer as owner and delegate.
- Register the pausers.
- Wire each remote: peer, libraries, executor and ULN configurations, and enforced options.
- Set the rate limits and their accounting types: per-EID for each remote (usually zero until a governance spell activates the route), and on mainnet also global across all remotes.
- Hand over to governance: `MCD_PAUSE_PROXY` on mainnet, `gov` (the `L2GovernanceRelay`) on remote chains.

A lockbox is deployed once per token, and remotes added later are wired to it by governance spells.

3.3 SSR Bridging

`L1SsrBridgeDeployer` and `L2SsrBridgeDeployer` deploy the SSR bridge, which carries the sUSDS rate data from an `SSROracleForwarderLZ` on mainnet through an `LZComposeReceiver` to an `SSRAuthOracle` on a remote chain. The forwarder and the receiver hold each other's address as an immutable, so the bridge is deployed in three steps:

1. `L2SsrBridgeDeployer` is deployed on the remote chain. Its constructor deploys the `SSRAuthOracle` , sets `maxSSR` , grants `DATA_PROVIDER_ROLE` to the address the receiver will be deployed at, grants `DEFAULT_ADMIN_ROLE` to `oracleAdmin` unless it is zero, and renounces its own admin role. The receiver

address is the deployer's `CREATE` address at nonce 2, since a contract's nonce starts at 1 and the oracle consumes it. It is exposed as `predictedReceiver`.

2. `L1SsrBridgeDeployer` is deployed on mainnet with `predictedReceiver` as the peer. It deploys the `SSROracleForwarderLZ`, sets the peer, the send library, the executor and ULN configurations, and enforced options with `IzReceive` and `compose gas`, then makes `MCD_PAUSE_PROXY` the delegate and owner.
3. The account that deployed `L2SsrBridgeDeployer` calls `deployReceiver()`, once. It deploys the `LZComposeReceiver` with the forwarder as source and the oracle as target, sets its receive library and ULN configuration, and makes the given `gov` its delegate and owner.

A governance spell then verifies and activates the forwarder.

3.4 Changelog

In `Version 2`, the deployer contracts are unchanged. Sky addressed the reported issues in the tests and the example scripts, and extended the README.

4 Trust Model

The deployer contracts hold no role once they have run. The only exception is `L2SsrBridgeDeployer` until `deployReceiver()` is called. Every OApp is handed to governance. Trust therefore sits with the account that deploys and parameterizes the deployers, with the governance contracts that receive the OApps, and with the LayerZero infrastructure the OApps are configured to rely on. For the roles on the deployed OApps themselves, we refer the reader to the respective audit reports.

4.1 Roles

Governance (`MCD_PAUSE_PROXY` on mainnet, the `L2GovernanceRelay` on remote chains)

- Trust level: Fully trusted.
- Permissions: Owner and endpoint delegate of every deployed OApp. Can upgrade the OFT adapters, change their peers, libraries, DVN configurations, rate limits and pausers, unpause them, and migrate the tokens locked in the lockbox. The `L2GovernanceRelay` additionally owns the governance receiver and changes its own parameters through the actions it executes.
- Worst-case impact: Upgrade the lockbox and take all tokens locked on mainnet, or mint unbacked tokens on a remote chain through an adapter that holds mint authority.

Deployer

- Trust level: Untrusted.
- Permissions: Chooses every constructor argument: the token, the remote peers and their configurations, the rate limits and accounting types, and the pausers. On remote chains, it also chooses the endpoint and the governance recipient. For the governance bridge, it chooses the mainnet sender and relay, the delay, the grace period, and the cancellers. The deployer contracts are permissionless, so anyone can deploy them. The account that deployed `L2SsrBridgeDeployer` additionally calls `deployReceiver()` once, and chooses the forwarder the receiver accepts messages from, its receive configuration and its governance recipient.
- Worst-case impact: A deployment parameterized with an attacker-controlled peer, forwarder, governance recipient or pauser, a weak DVN configuration, or a wrong mainnet relay gives the attacker control over the OApp or lets forged messages through. Once a spell activates it, this leads to unbacked minting on the remote chain, a drain of the lockbox up to its rate limits, forged governance actions on the remote chain, or arbitrary data in the SSR oracle.

Governance must verify every deployment in full before a spell onboards it. The activation spells check only part of the deployed configuration, so the remainder has to be verified off-chain.

End Users

- Trust level: Untrusted.
- Permissions: None on the deployer contracts.
- Worst-case impact: None is assumed.

4.2 External Dependencies

LayerZero

- Trust level: Semi-trusted.

- Assumptions: The endpoints, message libraries and executors work correctly as documented. On remote chains, the given endpoint is the canonical LayerZero `EndpointV2`. The security of every route rests on the DVN set passed to the deployer. DVNs able to reach the configured threshold can forge messages. On a token route, they can mint on the remote chain or release locked tokens on mainnet up to the rate limits. On the governance and the SSR oracle routes, the LayerZero-aligned DVNs alone cannot reach the threshold, so forging a message also requires the CCIP wing or the Sky multisig.

4.3 General Assumptions

- The deployed contracts and their dependencies work correctly as documented.
- The bridged tokens are standard ERC20 tokens, and their mint and burn authority on remote chains is granted only to the adapter and the relay.
- `L2SsrBridgeDeployer` is deployed on a chain with standard EVM `CREATE` address derivation, so that the receiver lands at `predictedReceiver`.
- Each deployment is verified against the intended parameters before a spell onboards it.

5 System Considerations

We leverage this section to highlight findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

SC1 Deployment and Validation

System Consideration	Version 1
----------------------	-----------

The deployers perform no sanity checks of their own, and the activation spells verify only parts of a deployment. Governance must therefore ensure that each deployment and its configuration are fully validated, either off-chain or by the activation spell, before onboarding it.

The checks can be outlined as follows:

- The LayerZero configuration of the deployed contracts, for example the peers, the send and receive libraries, the executor and the DVN configurations.
- The contract-specific configuration, for example `delay`, `gracePeriod` and `bud` of the `L2GovernanceRelay`, `maxSSR` and `oracleAdmin` of the `SSRAuthOracle`, or the pausers, accounting types and rate limits of the OFT adapters.

Several of these accept values with a silent default meaning. For example, a relay `delay` of 0 removes the cancellation window, and a `maxSSR` of 0 leaves the oracle uncapped. Others can have unexpected consequences. For example, with an empty `bud`, governance messages could never be skipped, which could hypothetically lead to a DoS of the remote chain's governance. Similarly, an OFT route deployed with non-zero per-remote rate limits is live at handoff, so `activateOft()`, which requires them to be zero, can no longer verify it.

The deployment and activation process should be validated as well, as the ordering of operations matters. For example, if a route is activated on mainnet immediately while its remote side is activated through a relayed spell, the route is open on mainnet before the remote side accepts transfers. Transfers sent in this window lock tokens in the lockbox and revert on the remote chain. They can be retried once the remote side is active, but users bridging during that window are inconvenienced. The sequence for a new chain described in the README of **Version 2** follows this pattern: the L1 spell wires the existing lockboxes with their intended rate limits, while the new adapters are opened by `LZL2Spell.activateOft` relayed through `relayToL2`, which the `L2GovernanceRelay` executes only after its `delay`.

Note that the deployers deliberately support various configurations (for example, a route that is live at handoff and validated off-chain instead of through `activateOft()`). Which configuration is intended, and how it is validated, has to be decided per deployment.

SC2 Gas Values After the Amsterdam Hard Fork

System Consideration	Version 1
----------------------	-----------

The Amsterdam hard fork (EIP-8037, EIP-8038) makes state writes much more expensive. A new storage slot costs about 108k gas (97,920 state gas plus 10,000 execution gas) instead of 22k. An update costs 10,000 instead of 2,900. Under EIP-8037, a gas cap set with `{gas: N}` must also pay this state gas if the transaction has no state-gas reservoir. We assume that executors and CCIP transmitters do not send transactions above

16.7M gas, so there is no reservoir. The numbers are based on the EIPs, which are in Last Call. They apply on a chain once it activates the fork.

- **OPTIONS_GAS (SSR forwarder, lzReceive) must be increased.** It is 100,000. After the fork, a single SSTORE to a new slot already costs more than this. Every `lzReceive` creates at least one new slot: the compose-queue entry in the endpoint. **Consequence:** a `refresh()` that does not add `lzReceive` gas through its options fails at delivery. The message stays verified and anyone can retry it with more gas. Until then, the remote oracle is not updated.
- **COMPOSE_GAS (SSR forwarder, lzCompose) is enough.** It is 200,000. Each `lzCompose` does two SSTOREs: it marks the compose-queue entry as delivered and it writes the oracle data. Both update existing slots, so they cost 10,000 each and no state gas. Only the first oracle write creates a new slot.
- **The CCIP adapter's dstConfig.gas must be increased.** It was 200,000 in **Version 1** and is 400,000 in **Version 2**. One CCIP message makes the broadcaster verify all 4 replicas, and each replica creates a new slot in the receive library. These 4 new slots alone cost about 432,000 gas, so 400,000 is still not enough. **Consequence:** CCIP execution fails, the CCIP wing does not vote, and someone must execute the message manually and pay for it. The LayerZero-aligned DVNs and the multisig can still reach the 8 of 15 threshold.
- **OFT options gas must be measured.** This is the options gas of the OFT routes to and from Ethereum. It was 130,000 in **Version 1** and is 200,000 in **Version 2**. A transfer to a new recipient creates at least one new balance slot. Today, the first inbound transfer on the mainnet lockbox needs about 147k gas, of which about 22k is for the new balance slot. After the fork, this slot alone costs about 108k, which suggests that 200,000 is not enough. **Consequence:** transfers may fail at delivery and need a manual retry.

In short: the fork changes the gas cost of message delivery. `OPTIONS_GAS` and the CCIP gas become too low, `COMPOSE_GAS` stays enough, and the OFT options gas must be measured. In **Version 2**, the README states that the gas values of the example scripts have to be re-measured for the chains and EVM version a deployment targets. Governance should keep these values in line with the forks on each destination chain. Then messages are still delivered automatically. A new value does not help messages that were sent before the change.

SC3 Initcode Size Limits the Number of Remotes

System Consideration Version 1

The OFT deployers take all remotes as a constructor argument, which is ABI-encoded into the initcode. Each `RemoteWiring` with a four-DVN configuration adds about 1,250 bytes to the `L10FTDeployer` creation code. Above roughly 16 remotes, the initcode exceeds the EIP-3860 limit of 49,152 bytes and the deployment transaction is invalid. Larger DVN sets lower that number.

Further remotes can still be wired later through a spell.

6 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

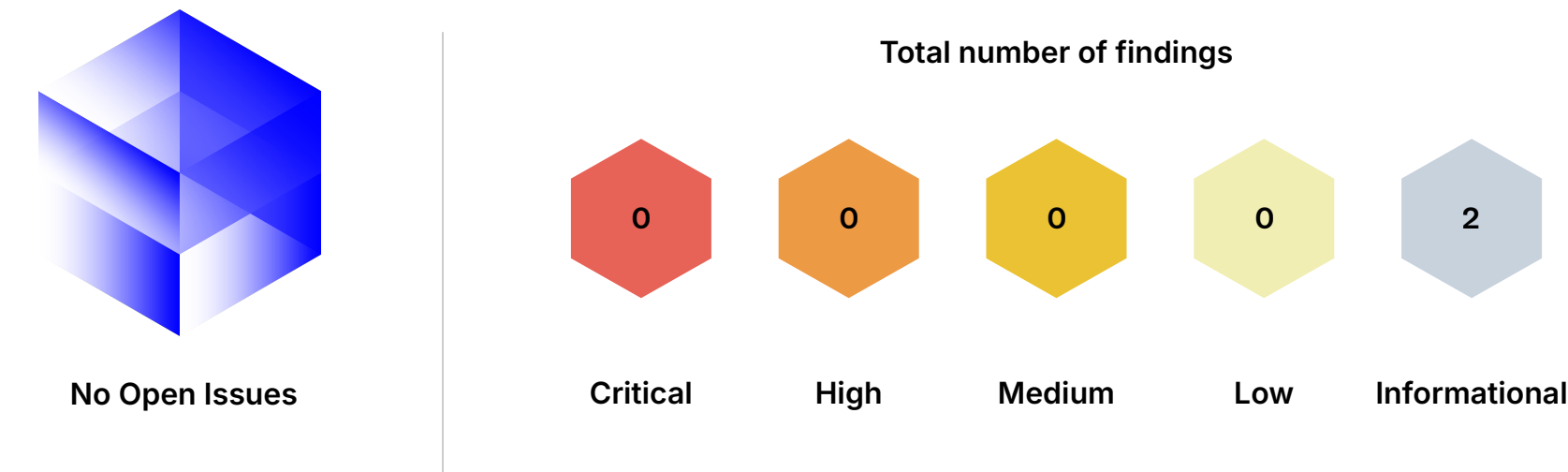
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

7 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



7.1 Overview

The following table lists all identified findings along with their severity and current status. A finding is considered resolved once it has been fully corrected or the specification has been changed accordingly. Non-informational findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Info	#001	First Inbound Transfer Exceeds the Enforced Receive Gas	Code Corrected	✓
Info	#002	OFT Routes Inherit LayerZero's Default Optional DVNs	Code Corrected	✓

7.2 Descriptions

#001 First Inbound Transfer Exceeds the Enforced Receive Gas

Informational Version 1

Code Corrected ✓

`L1OFTDeployer` and `L2OFTDeployer` set the enforced `lzReceive` gas of each route to `cfg.optionsGas`, and `LZInit.activateOft()` requires the activated route to keep exactly that value. The tests and examples use 130,000, the value of the V1 adapters. In addition to the per-EID buckets that the V1 adapters also update, the V2 lockbox updates a global rate-limit bucket.

On the mainnet lockbox, the first transfer on a route to a new recipient needs more than 140k gas. Executed with 130k gas, it reverts. Later transfers need less than 110k.

No funds are lost. Execution is permissionless, so anyone can retry the message with more gas, and senders can add gas through `extraOptions`. Until then, the first transfer on each route into mainnet is stuck.

Code Corrected

The tests and the example scripts now set the enforced `lzReceive` gas of the OFT routes to 200,000. This is above the roughly 147k gas the measured first inbound transfer on the mainnet lockbox. The README now states that the gas values of the examples have to be re-measured for the chains and EVM version a deployment targets. The value remains an input to the deployers, see also [Gas Values After the Amsterdam Hard Fork](#).

#002 OFT Routes Inherit LayerZero's Default Optional DVNs

Informational Version 1

Code Corrected ✓

The OFT DVN configurations in the tests and examples set `optionalDVNCount` to 0. In `UlnBase.getUlnConfig()`, 0 means DEFAULT, so the effective configuration takes the optional DVNs and threshold from LayerZero's default configuration for the route. LayerZero can change that default at any time.

Today the default optional set for Avalanche and Base is empty, so the effective set is exactly the four required DVNs. If LayerZero adds optional DVNs with a threshold to the default, every OFT message also needs that threshold of their signatures, and token transfers halt if they do not sign. Security is not reduced, because the required DVNs still have to verify.

Setting `optionalDVNCount` to `NIL_DVN_COUNT` (255) pins an empty optional set.

Code Corrected

The OFT DVN configurations in the tests and the example scripts now set `optionalDVNCount` to `NIL_DVN_COUNT` (255), with an empty optional set and a zero threshold. The effective configuration therefore no longer depends on LayerZero's default optional DVNs.

8 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.